Vol.2017-OS-141 No.21
2017/7/27

オンチップアクセラレータを用いたパイプライン並列処理のためのOS支援機構の検討

小柴 篤史[1,a)]　坂本 龍一[2]　並木 美太郎[1]

概要：オンチップアクセラレータを用いたパイプライン並列処理は，チップ間のデータ転送とタスク実行をオーバラップすることでアクセラレータの演算性能を活用できる．しかしアクセラレータやDMAの実行制御，同期制御を頻繁に行う必要があるため，従来のデバイスドライバを介したアクセラレータ制御ではユーザ/カーネル空間のコンテキストスイッチに起因するオーバヘッドを招き，処理性能が低減する課題がある．そこで本研究では新たなOSの資源管理手法として，パイプライン並列処理向けのアクセラレータの制御機構を提案する．提案手法は，本来ユーザプロセスが行う煩雑なハードウェア制御をカーネルプロセスが代行する．これによりユーザ/カーネルプロセス間の通信を抑制し，制御オーバヘッドを削減する．本研究ではその初期検討として提案するOS機構をLinuxに実装し，ヘテロジニアスマルチコアプロセッサのプロトタイプを用いて評価した．アルファブレンダのパイプライン並列プログラムに提案機構を適用した結果，提案機構はデバイスドライバを用いる場合と比較してソフトウェアの制御オーバヘッドを86.2%削減し，プログラムの実行速度を1.66倍高速化することを明らかにした．

キーワード：ハードウェアアクセラレータ，パイプライン並列処理，ヘテロジニアスコンピューティング，オペレーティングシステム

1. はじめに

LSIプロセス技術の発展によってトランジスタの集積度の向上が進む一方，CPUコアの処理速度の向上はプロセス微細化に伴う消費電力，発熱の増加によって限界となっている．そこで，汎用的なCPUコアだけでは困難となったエネルギー性能と処理性能の向上を達成するため，特定のアプリケーションに特化したハードウェアアクセラレータが注目されている．アプリケーションの実行タスクに適したアクセラレータに処理をオフロードすることで，汎用的なマルチコアCPUと比較して数百倍のエネルギー効率を達成できる [1], [2]．このようなアクセラレータを活用するため，汎用CPUコアに加えて様々な種類のアクセラレータをオンチップで搭載したヘテロジニアスなSystem-on-Chip (SoC) が提案されている [3], [4], [5]．

オンチップアクセラレータへのタスクオフローディングはオフチップの共有メモリとのデータ転送を要するため，このデータ転送を高効率に制御することが重要となる [6]．アクセラレータと共有メモリ間のデータ転送を高速化する手法として，パイプライン並列処理が有効である [7]．パイプライン並列処理では，Direct Memory Access (DMA) を用いてDRAMとアクセラレータ間でデータを受け渡しつつ，アクセラレータによる演算を実行する．これによりデータ転送と演算をオーバラップすることでデータ転送時間を隠ぺいし，処理時間を短縮できる．このパイプライン並列処理はJPEGデコーダなどの複数のタスクで構成されるストリーミングアプリケーションに特に有効である．この場合，各タスクの実行に適した複数のアクセラレータを組み合わせてパイプラインを構築し，アクセラレータのローカルメモリ間でDMAを用いて中間データを直接受け渡す．これによりオフチップのメインメモリへのアクセスを抑制し，データ転送スループットを向上できる．

パイプライン並列処理では，従来のデバイスドライバを介したユーザプロセス主体のアクセラレータ制御がボトルネックとなる．アクセラレータは一般的な汎用OSでは外部デバイスとして扱われ，ベンダから提供されるデバイスドライバを用いて制御される．そのため，パイプライン実行中はユーザプロセスが各デバイスに対応するドライバを介してアクセラレータとDMAの実行制御，同期制御を行う必要がある．このユーザプロセスとデバイスドライバ間の通信はコンテキストスイッチを引き起こし，これに起因

[1] 東京農工大学
[2] 東京大学
[a)] koshiba@namikilab.tuat.ac.jp

するオーバヘッドは数 10 マイクロ秒にのぼる．パイプライン実行では全てのデータを処理するまで各ハードウェアの実行制御と同期を繰り返し行う必要があるため，頻繁なハードウェア制御のオーバヘッドよってアクセラレータの処理性能が低減してしまう．

そこで提案手法では，アクセラレータを用いたパイプライン並列実行をサポートする OS の新たな機能として Pipeline Parallelism Manager (PPM) を提案する．PPM はカーネル空間で動作し，本来はユーザプロセスがパイプライン実行中に行うハードウェア制御を代行する．カーネル空間から直接アクセラレータ/DMA の実行制御や同期を行うことで，ユーザ/カーネル間通信を抑制し，ソフトウェア制御によるオーバヘッド時間を削減する．PPM がユーザプロセスの要求に応じてパイプライン実行を代行するため，ユーザプロセスは実行開始時に利用するアクセラレータ，実行タスクの依存関係，演算対象のデータに関する情報を実行リクエストとして PPM へ渡す．ユーザプロセスは直後にスリープし，PPM は全てのデータを処理するまで与えられた情報に基づいてパイプライン実行を行い，終了時にユーザプロセスを起床させる．これにより，パイプライン実行中の煩雑なユーザ/カーネル間通信を排除する．

提案手法の効果を検証するため，本研究では画像処理アクセラレータの実チップを用いたヘテロジニアスマルチコア環境のプロトタイプを構築し，提案する PPM を Linux のカーネルモジュールとして実装した．初期評価としてアクセラレータを用いた 3 段パイプライン並列プログラムの実行時間を計測し，提案手法をデバイスドライバによる制御と比較した．評価結果は，提案手法はデバイスドライバによる制御時と比較して 86.2%のソフトウェア制御オーバヘッドを削減し，プログラムの実行速度を 1.66 倍高速化することを示している．

2. パイプライン並列制御の課題

2.1 アクセラレータのパイプライン並列制御

本節では，本論文で対象とするヘテロジニアスマルチコア環境と，パイプライン並列制御の概要について述べる．近年の計算機に対する演算性能の要求の高まりによって，様々な種類のアクセラレータが提案されており，それらのアクセラレータを CPU コアと同一のチップ上に搭載したヘテロジニアスなマルチコア SoC が注目されている [3], [4], [8]．図 1 に本研究の対象とするヘテロジニアスマルチコア環境の概要図を示す．本環境では同一のダイ上に複数の CPU コアおよびアクセラレータが搭載されており，個々のコアは Network-on-Chip で接続されている．個々の CPU コアとアクセラレータは少量のプライベートなキャッシュおよびローカルメモリを持ち，大容量のメインメモリ (DRAM) を共有する．共有メインメモリとアクセラレータのローカルメモリ間のデータ通信はホスト CPU が DMA

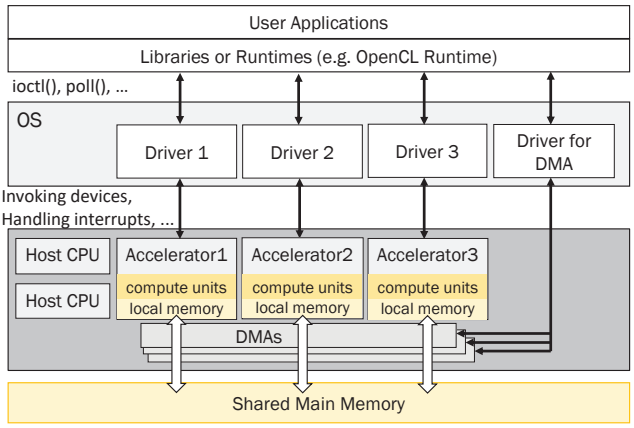

図 1 ヘテロジニアスマルチコア環境

を用いて行う．ユーザアプリケーションは，計算量の多い処理を適切なアクセラレータにオフロードすることで，タスク実行を高速化できる．

アクセラレータへのタスクオフローディングは処理時間の短縮およびエネルギー削減に有効だが，共有メモリとアクセラレータ間のデータ転送制御が課題となる．近年のアプリケーションはアクセラレータのローカルメモリよりもはるかに大きいデータセットを扱うものが多いため，データを全て処理するまでに共有メモリとのデータ転送を頻繁に行う必要がある [8]．また，JPEG デコーダなどの異なるタスクで構成されるストリーミングアプリケーションを複数のアクセラレータを用いて高速化する際，中間データを共有メモリを介して受け渡すとスループットの小さいオフチップの共有メモリへのアクセスが増加し，バンド幅が不足する．このオフチップメモリとのデータ転送がボトルネックとなり，アクセラレータの処理性能を低減させる．

オンチップアクセラレータのデータ転送を高効率に行うには，パイプライン並列処理が有効である．図 2 にパイプライン並列処理の概要図を示す．パイプライン実行では，タスクをオフロードするアクセラレータとデータ転送用の DMA でパイプラインを構築する．データをアクセラレータで一度に処理可能なサイズに分割し，分割したデータをパイプライン式に順に処理していく．パイプライン実行における演算単位をパイプラインステージと呼び，アプリケーションは各ステージの開始/終了時にアクセラレータの初期化と同期処理を行う．アクセラレータ上での演算と入出力データの転送をオーバラップすることで，データ転送のオーバヘッドを隠ぺいし，高速にタスクを処理できる．また，アクセラレータにオフロードするタスクの数に応じてパイプラインの段数を増やすことで，ストリーミングアプリケーションの各タスクを並行して実行でき，演算時間を短縮できる．パイプライン並列ではアクセラレータ間での中間データの受け渡しを DMA を用いて共有メモリを介さずに直接行うことで，低速なオフチップメモリへのアクセスを低減できる．

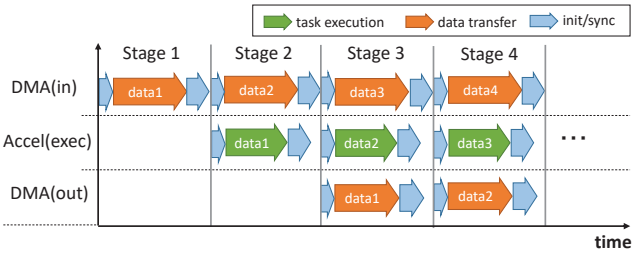

図 2 パイプライン制御の実行フロー (アクセラレータ数 1，パイプライン段数 3 の場合)

2.2 デバイスドライバの制御オーバヘッド

前述のパイプライン並列処理はアクセラレータのデータ転送スループットを向上できる一方で，従来のデバイスドライバを介したハードウェア制御のオーバヘッドが課題となる．図 2 に示すように，パイプライン並列実行では，パイプラインステージごとに各アクセラレータと DMA の初期化，同期処理を行う必要がある．オンチップアクセラレータでは回路面積や消費電力の制約によってローカルメモリサイズが数 kB から数 10kB に制限されるため [8], [4]，1 ステージ当たりの演算時間は短くなり，実行時の頻繁なハードウェア制御が必要となる．一方，これらのハードウェアの初期化や同期処理などの資源管理は，従来の OS ではデバイスドライバによって実現されている．デバイスドライバは個々のアクセラレータごとに提供され，ユーザアプリケーションはデバイスドライバに ioctl() や poll() などのシステムコールを発行することで対応するアクセラレータを制御する．しかしこのデバイスドライバを介した制御によって生じるユーザ/カーネル間通信はマイクロ秒オーダーのオーバヘッドを要し，アクセラレータの処理性能を低減させる．

図 3 にデバイスドライバを用いたアクセラレータ制御の流れを示す．ユーザプロセスはデバイスドライバに対して ioctl() を発行し，デバイスドライバを呼び出す．デバイスドライバは対応するデバイスの処理を開始し，ユーザプロセスは実行が完了するまでスリープする．実行終了後，デバイスが発行した割り込みを割り込みハンドラが検知して，ユーザプロセスを復帰させる．ここで，ioctl() によるユーザプロセスからのデバイスドライバの呼び出しにかかる時間 (T(ioctl))，および割り込みハンドリングからユーザプロセスの起床までの時間 (T(wakeup)) がユーザ/カーネル通信のオーバヘッドとなる．

このデバイスドライバの制御オーバヘッドが処理性能に与える影響を検証するため，4 章で後述するヘテロジニアスマルチコア環境のプロトタイプを用いて，アクセラレータ上で 24bit アルファブレンダを実行した時のデバイスドライバのオーバヘッドを計測した．図 4 に計測した個々の処理の実行時間を占めす．図 4 に示すように，T(ioctl) は約 4.0μs，T(wakeup) は約 25.6μs となり，ハードウェアの

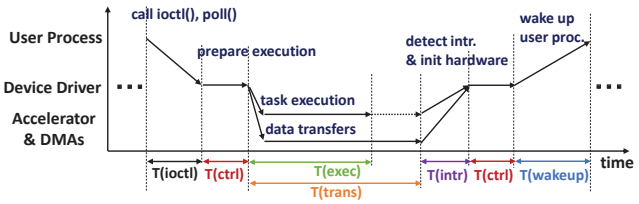

図 3 デバイスドライバを用いたアクセラレータ制御の流れ

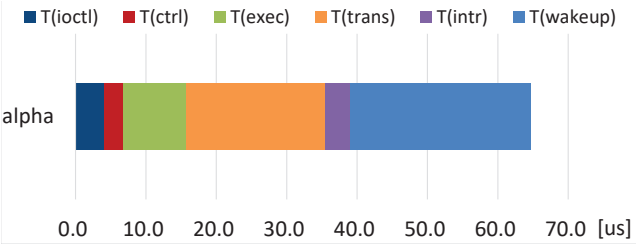

図 4 図 3 における各処理の実行時間．4 章で示すヘテロジニアスマルチコア環境を用いて計測．

初期化および同期処理そのものにかかる時間 (T(ctrl)) と比較して大きなオーバヘッドとなる．このオーバヘッドはパイプラインの段数が増え，制御すべきデバイスドライバが増えるにつれて大きくなると予想されるため，パイプライン並列処理におけるボトルネックとなる．そこで本研究では，このユーザ/カーネル間通信に起因するオーバヘッドを削減し，アクセラレータを用いたパイプライン並列処理を高効率に行うための OS 支援機構を提案する．

3. パイプライン並列制御を代行する OS 機能

本研究では，マルチコアアクセラレータを用いた細粒度なパイプライン並列処理を対象として，アクセラレータや DMA の実行時制御を低オーバヘッドで可能にする OS の資源管理機構として PPM(Pipeline Parallelism Manager) を提案する．本章では提案する PPM の概要について述べる．

3.1 全体構成と概要

PPM はカーネル空間で動作し，本来ユーザアプリケーションが担うパイプライン処理中のアクセラレータ制御，DMAC 制御を代行する．PPM はパイプライン並列プログラムが利用するアクセラレータと DMA の実行制御をユーザプロセスを介さずにカーネル空間から直接行う．これにより，パイプライン並列処理中にユーザプロセスと OS 間の通信によって生じるオーバヘッド時間を削減し，アクセラレータの処理性能を向上する．

図 5 に PPM のシステム構成を示す．PPM は，主に以下の三つの要素で構成され，個々が提供する機能を用いてユーザプロセスを介さずに様々なアクセラレータを用いたパイプライン並列制御を実現する．

- パイプライン実行情報 (papp info):

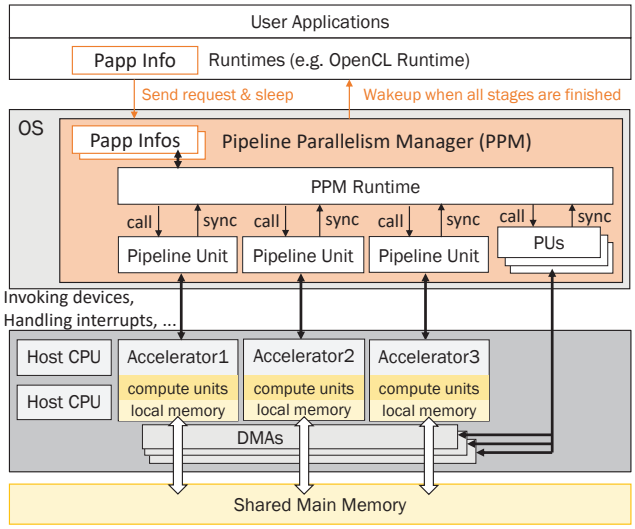

図 5 パイプライン並列制御機構の構成

パイプライン並列プログラムの実行時のアクセラレータ制御に必要な情報 (使用するアクセラレータ，パイプライン段数，データサイズと場所，各ステージにおけるアクセラレータの処理内容など)．PPM の起動時にユーザアプリケーションによって渡される．

- PPM Runtime:
  ユーザアプリケーションから与えられたパイプライン実行情報に基づき，Pipeline Unit を介してアクセラレータを制御する．パイプラインステージごとのアクセラレータと DMA の実行制御，同期制御を担う．
- Pipeline Unit:
  各アクセラレータ/DMA の実行時制御においてハードウェアの種類に依存する処理 (初期化，割り込みハンドラ等) を共通のインタフェース (PU コマンド) として PPM Runtime に提供する．

PPM を用いたパイプライン並列処理において，まず最初にアプリケーションはパイプライン実行情報を ioctl() で PPM へ通知する．PPM はアプリケーションのパイプライン実行情報を参照して実行に必要なアクセラレータと DMA を確保し，PPM Runtime が各デバイスを初期化して実行を開始する．PPM Runtime は全てのデータを処理するまでパイプラインステージごとに各デバイスの実行制御や同期制御を Pipeline Unit を介して行う．全てのパイプラインステージの実行が完了したのち，PPM はユーザアプリケーションを起床させ，処理を終える．ユーザプロセスと PPM 間の通信は実行開始時と終了時のみ行われるため，デバイスドライバを用いる場合と比較して制御オーバヘッドを削減できる．次節より，PPM を構成する各機能の詳細を示す．

3.2 パイプライン実行情報

PPM がカーネル空間からユーザアプリケーション実行中のアクセラレータ制御を代行するには，アクセラレータに割り当てるタスクの依存関係や演算データなどの情報を PPM が事前に知る必要がある．そこで提案手法では，この実行時のアクセラレータ制御に関する情報をパイプライン実行情報としてユーザアプリケーションが生成し，実行開始時に PPM に与える．パイプライン実行情報は主に下記の情報で構成される．

- ハードウェア情報:
  パイプラインを構成するアクセラレータの数と種類
- タスク情報:
  各アクセラレータに割り当てるタスクとタスク間の依存関係
- データ情報:
  入出力データのメモリ領域とデータサイズ，1 ステージ中に処理するデータ単位

パイプライン実行情報は，OpenCL などの並列演算ライブラリのランタイム機能がユーザアプリケーションの API コールを監視，解析することで動的に生成する．これにより，ユーザはアプリケーションを変更することなく PPM を利用できる．パイプライン実行情報の動的生成方法の詳細の検討については，今後の課題である．

PPM はマルチプロセスからの複数のパイプライン実行要求に対応するため，キューを用いて各パイプライン実行情報を保持，管理する．各アプリケーションのパイプライン実行情報はリクエストキューに保持され，後述の PPM Runtime によって実行可能なものから処理される．これによりマルチタスク間でのアクセラレータ共有を実現する．

3.3 PPM Runtime によるアクセラレータ制御

PPM Runtime はパイプライン実行情報に基づき，各アクセラレータ上でのタスク実行，同期制御を行う．最初に PPM Runtime はリクエストキューを参照してパイプライン実行情報を読み取り，アクセラレータの利用状況を調べ，各アプリケーションが実行可能かどうか調べる．アクセラレータが既に他のアプリケーションによって利用されていた場合，アプリケーションを実行待ちとする．アクセラレータが利用可能な場合は実行情報に基づきパイプライン実行を開始する．

実行時，PPM Runtime は最初にパイプライン実行情報に基づいてアプリケーションが利用するアクセラレータと DMA の確保および初期化処理を行い，各パイプラインステージの実行を開始する．PPM Runtime は各ステージの最初にアクセラレータと DMA で実行を開始し，全てのアクセラレータおよび DMA の実行が完了したら，同様に初期化と次のステージの実行を開始する．これを全ステージの実行が終わるまで繰り返し行う．PPM Runtime は全ての処理が終了したらユーザプロセスを起床させ，対応するパイプライン実行情報を破棄し，リクエストキューを参照

表 1　PU コマンド

コマンド	アクセラレータ	DMA
init	実行タスクの設定	入出力データの設定
exec	タスク実行開始	データ転送開始
done	割り込み検知時の初期化，終了通知 (sync)	

して次のアプリケーションの実行を開始する．

3.4 Pipeline Unit によるアクセラレータの抽象化

PPM Runtime はアプリケーションの要求に応じて，多様なアクセラレータを組み合わせてパイプラインを構築し，各アクセラレータの実行制御を行う必要がある．アクセラレータのハードウェア依存性を吸収し，汎用性を保証するため，提案手法ではアクセラレータ固有の処理を PPM Runtime が利用可能な API(PU コマンド) として抽象化し．それらの API 群を Pipeline Unit として管理する．PPM Runtime が個々のアクセラレータに対応する PU コマンドを用いてアクセラレータを制御することで，種類の違いを意識せずに各アクセラレータを共通のインタフェースで扱うことができる．

Pipeline Unit はアクセラレータの開発ベンダによってカーネルプロセスが利用可能なライブラリとして提供される．Pipeline Unit はシステム内の全てのアクセラレータ，DMA に提供され，一つのアクセラレータにつき一つの Pipeline Unit が対応する．表 1 に Pipeline Unit が提供する PU コマンドを示す．Pipeline Unit は，パイプライン実行中のアクセラレータおよび DMA の処理 (初期化，演算/データ転送実行，割り込みハンドリング) を PU コマンドとして提供する．PPM Runtime はパイプライン実行情報に基づき，PU コマンドを用いて各アクセラレータ，DMA をパイプラインステージごとに制御する．Pipeline Unit を介した制御によって，PPM Runtime は異なる種類のアクセラレータを用いたパイプライン実行を実現する．

4. 実装

提案手法が実システムに適用可能であることを示すため，本手法を Linux に実装し，実際のヘテロジニアスマルチコア環境で評価する．表 2 に開発評価環境の詳細を示す．

まず，提案手法の開発および評価環境として，Zynq の ARM プロセッサと画像処理アクセラレータである CC-SOTB [9] で構成されるヘテロジニアスマルチコアプロセッサのプロトタイプ環境を構築した．図 6 にプロトタイプ環境の構成を示す．本環境では，MicroZed ボードの Zynq プロセッサが持つ Cortex-A9 をホスト CPU，実シリコンチップとして実装される CC-SOTB をアクセラレータとして用いる．MicroZed と CC-SOTB チップは専用のマザーボード上に接続され，MicroZed の Zynq プロセッサと CC-SOTB は Zynq FPGA に実装した通信バスを介して通信する．

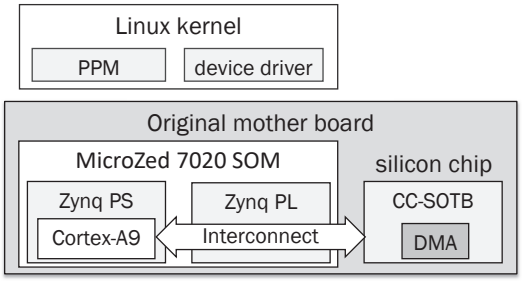

図 6　ヘテロジニアスマルチコアプロセッサのプロトタイプ環境

表 2　開発，評価環境

項目	詳細
評価用ボード	MicroZed SOM 7020
ホスト CPU	ARM Cortex A9
アクセラレータ	CC-SOTB (3kB local memory) [9]
チップ間通信	3Gbps Network-on-Chip [10]
ホスト OS	Linux 4.4.0

このヘテロジニアスマルチコア環境を対象に，提案するパイプライン並列制御機構を Linux のカーネルモジュールとして実装した．Linux は MicroZed の Cortex-A9 上で動作する．CC-SOTB のメモリやコントロールレジスタは Linux のアドレス空間にメモリマップされており，ソフトウェアは CPU の load/store 命令を用いてアクセスできる．

5. 評価

本章では，前節で述べたヘテロジニアスマルチコア SoC のプロトタイプ環境を用いて提案手法の基礎評価を行う．

5.1 評価方法

提案手法がパイプライン並列制御の高速化に有効であることを示すため，提案手法を CC-SOTB アクセラレータに適用し，パイプライン処理を行うベンチマークを実行した時の処理時間を評価した．評価用ベンチマークとして，640 × 480 ピクセルのカラー画像データ (約 922kB) に対して 24bit アルファブレンダを実行するプログラムを実装した．本プログラムにおける 1 ステージ当たりの演算データサイズは約 1.5kB，パイプライン段数は 3 段 (DMA × 2, CC-SOTB)，パイプラインステージ数は 612 ステージである．本評価では，既存のデバイスドライバを用いた資源管理手法に対する提案手法の効果を明らかにするため，ユーザプロセスがデバイスドライバを介して CC-SOTB を制御した場合と，提案手法を用いてカーネルモジュールが直接制御した場合を比較する．本実験では，ホスト CPU の動作周波数を 667MHz，アクセラレータの動作周波数を 60MHz に設定した．プログラムの実行時間は ARM プロセッサに搭載されているタイマ機能を用いて実測した．

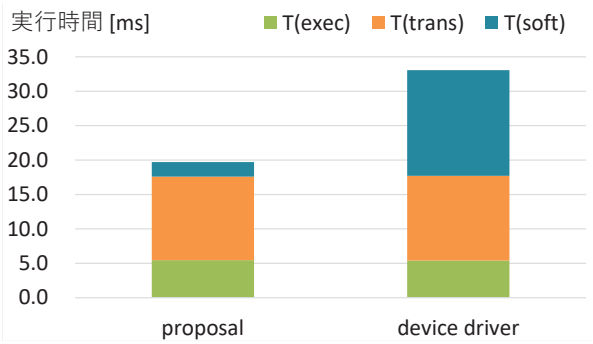

図 7 24bit アルファブレンダの実行時間．T(exec) は CC-SOTB の演算時間，T(trans) は DMA によるデータ転送時間，T(soft) はソフトウェアによる制御オーバヘッドを表す．T(exec) と T(trans) はオーバラップしている．

5.2 評価結果と考察

図 7 に提案手法 (proposal) とデバイスドライバ (device driver) による CC-SOTB を用いた 24bit アルファブレンダの実行時間を示す．図 7 より，デバイスドライバによる制御では実行時間が 32.9ms なのに対し，提案手法では 19.8ms と，処理速度を約 1.66 倍向上していることが分かる．この要因として，デバイスドライバでは 15.4ms がソフトウェア制御オーバヘッド (T(soft)) に費やされている一方，提案手法では 2.1ms と 86.2%削減していることが挙げられる．これは，デバイスドライバの方式において実行時間の多くを占めるユーザ/OS 間のコンテキストスイッチを提案手法ではほぼ排除できるためである．この結果は，提案手法がアクセラレータのパイプライン並列プログラムの処理性能向上に効果的であることを示している．

6. 関連研究

アクセラレータ制御においてユーザ/カーネル間通信のオーバヘッドを削減する手法として，ハードウェアベースの資源管理機構が多く提案されている．Cong らは，アクセラレータのタスク間共有や軽量な割り込みハンドリングをサポートするハードウェアコントローラを提案している [3]．Nachiappan らは，ソフトウェアを介さずにアクセラレータ間で直接データの受け渡しを可能にする通信プロトコルを提案している [11]．また，AMD が提供している Heterogeneous System Architecture (HSA) や，IBM が提案している CAPI プロトコルは，CPU と GPU/FPGA 間で仮想アドレス空間を共有することで，ユーザアプリケーションがアドレス空間を介して GPU や FPGA を直接制御可能にする [12], [13]．これらの手法はアーキテクチャやアクセラレータそのものを変更することで，アクセラレータ制御やアクセラレータ間のデータ通信制御における OS のオーバヘッドを削減している．一方，提案手法はシステムソフトウェア単体によるアプローチを取ることで，容易ではないハードウェア変更の必要なく高効率なアクセラレータ制御を実現する．

また，デバイスドライバに代わるアクセラレータ制御用のシステムソフトウェアの研究が行われている．Rossbach らは，GPU を CPU と同様の計算資源として OS がタスク割り当て等を行うための抽象化手法と，ストリーミングアプリケーション向けのプログラミングモデルを提案している [14]．しかし，この手法は GPU のみを対象としており，マルチコアアクセラレータの同時制御はサポートしない．Menychtas らは，マルチプロセス間でのアクセラレータのタスクスケジューリング手法を提案している [15]．しかし，この手法はマルチタスク間でアクセラレータを公平に共有する事を目的としており，パイプライン実行はサポートしていない．また，Asmussen らは，ヘテロジニアスなメニーコアプロセッサにおいて，演算コアのヘテロジニアス性の抽象化と高効率なコア間のデータ転送をサポートするマイクロカーネルベースの OS を提案している [5]．提案手法は Linux などのモノリシックカーネルの機能の一部として提供されることを想定している点が異なる．

我々が調査した限りでは，ハードウェア変更なしでオンチップのマルチコアアクセラレータを用いた高効率なパイプライン並列処理を実現するシステムソフトウェアの研究としては我々の手法は初である．

7. おわりに

本研究では，オンチップアクセラレータを用いたパイプライン並列制御において，低オーバヘッドなアクセラレータの実行時制御を可能にする OS 支援機能を提案した．提案手法は，従来ユーザプロセスがデバイスドライバを介して行うアクセラレータや DMA の実行時制御をカーネルプロセスが代行することで，実行時のユーザプロセス/OS 間のコンテキストスイッチを抑制し，アプリケーションの処理時間を削減する．提案手法を Linux のカーネルモジュールとして実装し，ヘテロジニアスマルチコアプロセッサのプロトタイプ環境で評価した結果，提案手法はデバイスドライバによるパイプライン実行と比較して処理性能を 1.66 倍向上をすることを示した．

今後の課題としては，JPEG デコーダなどの複数のアクセラレータを用いたパイプライン並列処理に対する提案手法の性能評価が挙げられる．従来のデバイスドライバによる資源管理手法では，アクセラレータの数が増加するとソフトウェア制御オーバヘッドも増加すると予想される一方，提案手法はパイプライン段数に関わらず実行時のユーザ/カーネル間通信を省くことができるため，提案手法がより効果的に働くと考えられる．次に，提案手法を用いたプログラミングモデルおよび並列演算ライブラリとの連携方法の検討が挙げられる．一般的なヘテロジニアスコンピューティングでは，アプリケーションは OpenCL などの並列演算ライブラリを用いて記述され，実行時のアプリケーショ

ンの API コールに応じてランタイム機能がアクセラレータを制御する．ここで，並列演算ライブラリのランタイム機能がアプリケーションの挙動からパイプライン実行情報を動的に生成することで，ユーザはアプリケーションを変更することなく提案手法を利用できる．

謝辞 本研究は特別研究員奨励費 (16J06711) の助成を受けたものである．

参考文献

[1] Hameed, R., Qadeer, W., Wachs, M., Azizi, O., Solomatnikov, A., Lee, B. C., Richardson, S., Kozyrakis, C. and Horowitz, M.: Understanding Sources of Inefficiency in General-purpose Chips, *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, New York, NY, USA, ACM, pp. 37–47 (online), DOI: 10.1145/1815961.1815968 (2010).

[2] Liu, D., Chen, T., Liu, S., Zhou, J., Zhou, S., Teman, O., Feng, X., Zhou, X. and Chen, Y.: PuDianNao: A Polyvalent Machine Learning Accelerator, *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '15, New York, NY, USA, ACM, pp. 369–381 (online), DOI: 10.1145/2694344.2694358 (2015).

[3] Cong, J., Ghodrat, M. A., Gill, M., Grigorian, B. and Reinman, G.: Architecture Support for Accelerator-rich CMPs, *Proceedings of the 49th Annual Design Automation Conference*, DAC '12, New York, NY, USA, ACM, pp. 843–849 (online), DOI: 10.1145/2228360.2228512 (2012).

[4] Cota, E. G., Mantovani, P., Guglielmo, G. D. and Carloni, L. P.: An analysis of accelerator coupling in heterogeneous architectures, *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6 (online), DOI: 10.1145/2744769.2744794 (2015).

[5] Asmussen, N., Völp, M., Nöthen, B., Härtig, H. and Fettweis, G.: M3: A Hardware/Operating-System Co-Design to Tame Heterogeneous Manycores, *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '16, New York, NY, USA, ACM, pp. 189–203 (online), DOI: 10.1145/2872362.2872371 (2016).

[6] Chung, E. S., Milder, P. A., Hoe, J. C. and Mai, K.: Single-Chip Heterogeneous Computing: Does the Future Include Custom Logic, FPGAs, and GPGPUs?, *Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO '43, Washington, DC, USA, IEEE Computer Society, pp. 225–236 (online), DOI: 10.1109/MICRO.2010.36 (2010).

[7] Teimouri, N., Tabkhi, H. and Schirner, G.: Revisiting Accelerator-rich CMPs: Challenges and Solutions, *Proceedings of the 52Nd Annual Design Automation Conference*, DAC '15, New York, NY, USA, ACM, pp. 84:1–84:6 (online), DOI: 10.1145/2744769.2744902 (2015).

[8] Mantovani, P., Cota, E. G., Pilato, C., Guglielmo, G. D. and Carloni, L. P.: Handling large data sets for high-performance embedded applications in heterogeneous systems-on-chip, *2016 International Conference on Compliers, Architectures, and Sythesis of Embedded Systems (CASES)*, pp. 1–10 (online), DOI: 10.1145/2968455.2968509 (2016).

[9] Masuyama, K., Fujita, Y., Okuhara, H. and Amano, H.: A 297mops/0.4mw ultra low power coarse-grained reconfigurable accelerator CMA-SOTB-2, *2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, pp. 1–6 (online), DOI: 10.1109/ReConFig.2015.7393280 (2015).

[10] Miura, N., Koizumi, Y., Take, Y., Matsutani, H., Kuroda, T., Amano, H., Sakamoto, R., Namiki, M., Usami, K., Kondo, M. and Nakamura, H.: A Scalable 3D Heterogeneous Multicore with an Inductive ThruChip Interface, *IEEE Micro*, Vol. 33, No. 6, pp. 6–15 (online), DOI: 10.1109/MM.2013.112 (2013).

[11] Nachiappan, N. C., Zhang, H., Ryoo, J., Soundararajan, N., Sivasubramaniam, A., Kandemir, M. T., Iyer, R. and Das, C. R.: VIP: Virtualizing IP Chains on Handheld Platforms, *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ISCA '15, New York, NY, USA, ACM, pp. 655–667 (online), DOI: 10.1145/2749469.2750382 (2015).

[12] Kyriazis, G.: Heterogeneous System Architecture: A Technical Review, AMD (online), available from ⟨http://developer.amd.com/wordpress/media/2012/10/hsa10.pdf⟩ (accessed 2017-6-27).

[13] Stuecheli, J., Blaner, B., Johns, C. R. and Siegel, M. S.: CAPI: A Coherent Accelerator Processor Interface, *IBM Journal of Research and Development*, Vol. 59, No. 1, pp. 7:1–7:7 (online), DOI: 10.1147/JRD.2014.2380198 (2015).

[14] Rossbach, C. J., Currey, J., Silberstein, M., Ray, B. and Witchel, E.: PTask: Operating System Abstractions to Manage GPUs As Compute Devices, *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, New York, NY, USA, ACM, pp. 233–248 (online), DOI: 10.1145/2043556.2043579 (2011).

[15] Menychtas, K., Shen, K. and Scott, M. L.: Disengaged Scheduling for Fair, Protected Access to Fast Computational Accelerators, *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, New York, NY, USA, ACM, pp. 301–316 (online), DOI: 10.1145/2541940.2541963 (2014).